Configuring the Jtest Plugin for Maven

- Initial Setup
- Configuring Jtest Execution
  - Hierarchy
  - Configuring Analysis in the POM File
  - Configuring Analysis in the Command Line
- Manual Customization of Compilation Data in the POM File

Initial Setup

To integrate Parasoft Jtest with Maven, you need to modify the Maven settings in one of the following `settings.xml` files:

- `$M2_HOME/conf/settings.xml` - global settings located in the Maven installation directory
- `$HOME/.m2/settings.xml` - user settings located in the `.m2` folder the user's home directory

If both files exist, the user-specific will overwrite the global settings.

You can copy the global settings from your Maven installation to the `$HOME/.m2` directory, use it as a template, and modify according to the instructions below.

Certain tools, such as continuous integration (CI) servers and IDEs, may come with embedded Maven, which relies only on user settings and ignores global settings. In such cases, ensure that the user settings contain all modifications that are required for Jtest integration.

Configure the following settings in your `settings.xml` file:

- `<pluginRepository>` - specifies the path to the local Maven repository shipped with Jtest in [INSTALL_DIR]\integration\maven
- `<pluginGroups>` - specifies the appropriate groupids, artifactids, and goals to simplify calling the plugin from the command line
- the `jtest.home` property - specifies the path to the Jtest installation directory to simplify calling the plugin from the command line

Your settings.xml file may resemble the following:

```
<settings>
  <!-- ... -->
  <pluginGroups>
    <!-- ... -->
    <pluginGroup>com.parasoft.jtest</pluginGroup>
        <pluginGroup>com.parasoft.xtest.cbt</pluginGroup>
 </pluginGroups>

  <profiles>
    <!-- ... -->
    <profile>
      <id>jtest-settings-profile</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <jtest.home>PATH/TO/JTEST</jtest.home>
      </properties>
      <pluginRepositories>
        <pluginRepository>
          <id>jtest-local</id>
          <url>file://${jtest.home}/integration/maven</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <mirrors>
    <!-- ... -->
    <!-- prevention against mirrors with wildcard (*) matching -->
    <mirror>
      <id>jtest-local-mirror</id>
      <mirrorOf>jtest-local</mirrorOf>
      <!-- mirrors tag does not support properties -->
      <url>file://PATH/TO/JTEST/integration/maven</url>
          <!-- properties are not resolved in this tag -->
    </mirror>
  </mirrors>

</settings>
```

Configuring Jtest Execution

There are several ways to configure how Jtest analyzes and tests code:

- By configuring the plugin parameters directly in your project's `pom.xml` file
- In a `.properties` file provided with the `<settings>` or `<settingsList>` plugin parameters in the `pom.xml` file
- With the Maven command line options `-Djtest` or `-Dproperty`
- In the `jtestcli.properties` file in the Jtest installation directory or your HOME directory (see Configuration Overview)

Hierarchy

If you configure analysis with Maven and provide the user property as a variable (`${...}`), the following hierarchy will be applied:

- `-Djtest.[Maven property name]` (e.g., `-Djtest.settings="my.general.properties"`).
- the `pom.xml` file
- `-Dproperty.jtest.[property name]` (e.g., `-Dproperty.jtest.license.use_network=true`)

If the property is hardcoded, ie. provided as a real value in `pom.xml`, it has priority and cannot be overridden by command line settings. This results in the following hierarchy:

- the `pom.xml` file
- `-Djtest.[Maven property name]` (e.g., `-Djtest.settings="my.general.properties"`).
- `-Dproperty.jtest.[property name]` (e.g., `-Dproperty.jtest.license.use_network=true`)

For example, the test configuration specified in `pom.xml` as a user property `<config>${jtest.config}</config>` can be overridden with `-Djtest.config`. If it is hardcoded as `<config>builtin://Demo Configuration</config>`, it cannot be overridden.

The settings provided in the `jtestcli.properties` file always have the lowest priority.

Configuring Analysis in the POM File

You can configure analysis by providing appropriate plugin parameters directly in the `pom.xml` file of your project. See Jtest Goals Reference for Maven for the list of configuration parameters.

You can configure the Jtest Plugin for Maven as a regular Maven plugin, or as a reporting plugin.

> **Disable inheritance**
>
> By default, Jtest performs analysis of every project (module) when generating a report for the root project. For this reason, it is important to include the **`<inherited>false</inherited>`** tag in your POM to prevent redundant analysis of every nested module.

Your regular plugin configuration in the `pom.xml` file may resemble the following:

```
<project>
  <!-- ... -->
  <build>
    <!-- ... -->
    <plugins>
      <!-- ... -->
      <plugin>
        <groupId>com.parasoft.jtest</groupId>
        <artifactId>jtest-maven-plugin</artifactId>
        <version>1.2.15</version>
                <configuration>
          <settings>settings.properties</settings>
          <config>builtin://Recommended Rules</config>
          <resources>
            <resource>**/my/package/**/*.java</resource>
            <resource>**/*.xml</resource>
          </resources>
          <exclude>**/test/**</exclude>
          <report>report_dir</report>
          <publish>true</publish>
        </configuration>
      </plugin>

    </plugins>
  </build>
</project>
```

All the properties shown in the example are optional.

Configuring the Jtest Plugin for Maven as a Reporting Plugin

To enable the reporting plugin capability, configure the Jtest Plugin in the `<reporting>` tag. As a result, the Jtest report will be appended to the *Project Reports* section of the Maven Site generated after analysis.

The reporting plugin configuration in the `pom.xml` file may resemble the following:

```
<project>
  <!-- ... -->
  <build>
    <!-- ... -->
  </build>

  <reporting>
    <plugins>
    <!-- ... -->
      <plugin>
         <groupId>com.parasoft.jtest</groupId>
         <artifactId>jtest-maven-plugin</artifactId>
                <version>1.2.15</version>
                 <configuration>
            <config>builtin://Recommended Rules</config>
            <!-- ... -->
        </configuration>
        <inherited>false</inherited> <!-- IMPORTANT -->
      </plugin>

    </plugins>
  </reporting>
</project>
```

The following command line allows you to build modules and generate the site documentation pages:

```
mvn install site
```

The generated documentation is available in the root project directory $basedir/target/site.

See Usage section of the Apache Maven Site Plugin documentation for more information about site generation, staging, and deployment.

Configuring Analysis in the Command Line

You can configure analysis directly in the Maven command line by passing appropriate settings with the `-D` command line options:

- **`-Djtest.[setting]`** allows you to configure the settings dedicated for execution of the Jtest Plugin for Maven; see Jtest Goals Reference for Maven.
  Example: `mvn jtest:jtest -Djtest.config="builtin://Critical Rules"`
- **`-Dproperty.[setting]`** allows you to configure any of the Jtest settings (see Configuration Settings) provided as the KEY=VALUE pattern.
  Example: `mvn jtest:jtest -Dproperty.console.vebosity.level=high`

See Hierarchy for information about the hierarchical order the settings are applied.

Examples

Example 1. The following command line analyzes every module of the analyzed build using the default `builtin://Recommended Rules` test configuration:

```
mvn jtest:jtest
```

Example 2. The following analysis is configured with two .properties files passed with the `jtest.settingsList` option. The test configuration and the analysis scope are passed with `jtest.config` and `jtest.resource` respectively. Jtest will analyze the files of the "my.groupId:my-artifacId" project that match the pattern "/src/main/java/my/package/**/*.java".

```
mvn jtest:jtest
  -Djtest.config="builtin://Critical Rules"
  -Djtest.resource="my.groupId:my-artifacId/src/main/java/my/package/**/*.java"
  -Djtest.settingsList="project.specific.properties","/etc/jtest/license.properties"
```

Example 3. The following analysis is configured with the .properties file passed with the `jtest.settings` option. The test configuration is passed with `jtest.config`. The `property.console.verbosity` command line option overwrites the `console.verbosity` option specified in the .properties file.

```
mvn jtest:jtest
  -Djtest.config="builtin://Recommended Rules"
  -Djtest.settings="my.general.properties"
  -Dproperty.console.verbosity.level=high
```

Manual Customization of Compilation Data in the POM File

In rare cases, a large number of build plugins that perform compilation and highly customized Maven builds may lead to gaps or errors in the compilation data automatically detected by the Jtest Plugin for Maven. Jtest reports such issues in the Setup Problems section of the generated report.

If you verify that the detected compilation data contains errors, you can extend or override the data in the POM file of the analyzed module.

The following example shows customized compilation data of an analyzed Maven sumbodule. The configuration is extended with an additional classpath element, and specifies a new source level for the module. The compilation id is not specified, so the default compilation id will be used (**default-compile** is the default id for execution of maven-compiler-plugin).

```
<project>
  <!-- ... -->
  <build>
    <!-- ... -->
    <plugins>
      <!-- ...
        configuration with non default compiler plugin appending
        ${extra-classpath-element} and changing sourcelevel which
        cannot be detected automatically
      -->
      <plugin>
        <groupId>com.parasoft.jtest</groupId>
        <artifactId>jtest-maven-plugin</artifactId>
                <version>1.2.15</version>
 <configuration>
          <compilation>
            <!-- no id specified so Maven default: "default-compile" id will be used -->
            <classpath>
              <path>${extra-classpath-element}</path>
            </classpath>
            <sourcelevel>1.6</sourcelevel>
          </compilation>
        </configuration>
      </plugin>

    </plugins>
  </build>
</project>
```

If you need to modify compilation data for all projects, you can use the `-Djtest.dataUpdate` command line option.

See Compilation Data Model for more information about how to customize compilation data automatically detected by the Jtest plugins.